COMSC-210 Lab 1
Structs and Classes

GOOD PROGRAMMING PRACTICES *show / hide*

ABOUT THIS ASSIGNMENT
In this lab session, you will do a basic review of C++ `structs` and classes, including the writing of classes in separate *specification* and *implementation* files. The purpose is to prepare you for all of the work that is to come in this course, which rely heavily on your mastery of these C++ programming features. The labs in this assignment are based on chapter 1 of our Childs textbook.

In ALL lab work involving non-templated classes in H and CPP files (like those in lab 1), do NOT #include the class' CPP file in anything. Instead, include the CPP file in the compilation of the project, in addition to the CPP that contains `main`. Refer to the "Compiler Instructions" for Visual C++ or for Mac for how to compile "projects" involving multiple CPPs.

Never include `cout` statements in class functions unless it is part of the specification. You may use cout statements for purposes of debugging your code, but remove or deactivate them (and their includes) before final testing and submitting.

After you complete this lab assignment, post the required files to the COMSC server so that you receive the proper credit for this 50-point lab. Your posted files are due at midnight of the evening of the due date indicated in the course outline. Use the "Submit your FA2015 work" link on the class website to post your file for this lab, using the **lab1** folder provided.

LAB 1a: Class Programming And Testing `[ Road.h, Road.cpp, and RoadDriver.cpp ]`
Purpose. The purpose of this lab is for you to get practical experience with class programming, including the writing of a specification H file and an implementation CPP file, with appropriate testing with a driver CPP file.

Requirements. Write 3 files: **`Road.h`, `Road.cpp`, and `RoadDriver.cpp`**, to solve exercise #17 on page 23 of the Childs textbook. Note that the "asphalt" function is to be a member function of the class, and have only one parameter, and that asphalt thickness is NOT supposed to be a class member variable. Put the class specification in the H file, the implementation in `Road.cpp`, and `int main()` in `RoadDriver.cpp`. Include as much testing in the driver as you feel is necessary to assure that your class works right.

Store the width and length as data members in whatever units you like, but just make sure that the functions for setting and getting the length should be in *miles*, and the functions for setting and getting the width should be in *feet*. For the "asphalt" function, note that there should be only one parameter -- thickness *in inches*. That function should return volume in *cubic feet*. Do NOT store volume or thickness as data members!

In your driver, use assertions *and* print the expected results next to the actual results, so that your class' operation can be confirmed. Use a *calculator* to come up with your expected result *before* running any tests with your driver, so that you don't let the driver influence your expectations of it. If you use floating point variables in the class, test with floating point variables.

In your driver (and ALL the drivers you write for this course), test the H file's `#ifndef` by including it twice (the **"ifndef test"**). Also, create 3 Road objects -- an original and two copies. Using the original object, test ALL setters, and use the getters to verify that the setters work right. Then create the 2nd object as a *read-only* copy of the first -- test it (**"object copy testing"**). Then create the 3rd object as using assignment *after* declaration -- test it (**"object assignment testing"**). Prove that both copies are equal to the original, confirming that copies of your objects can be made successfully.

This kind of testing is for drivers only -- it is not for the "apps" that you will write in future labs. An app is a program that uses proven class files, presumably already tested.

DO NOT add class functions other than those specified -- no "constructors", no "overloaded operators".
DO use the "string buffer method" learned in lab 0c.
DO identify member functions as either getters or setters. If a function does not mutate any of the data members, then tag it with the "trailing `const`.
DO NOT remove the const keyword in the object copy testing. It's there to confirm that the getters are written as getters.
DO NOT put ifndef tests anywhere but a test driver CPP.
DO include the full set of identifying comments in EACH FILE as modeled in lab 0b -- not just the one line as modeled below.
DO include the full set of identifying cout statements as modeled in lab 0b, ONLY in the CPP with int main.

Submit the three files (2 CPPs and 1 H) to the class website for credit. Do NOT submit OBJ or EXE or any other executable file. And do NOT use file I/O -- all your input should be hard-coded, and all your output should be to `cout`.

Program I/O. Input: All hard-coded in the driver CPP only. (If you have a `cin` statement, then you did not do this right!) Output: From the driver CPP only, console (`cout`) output only.

Example.

Road.h	Road.cpp	RoadDriver.cpp
`// Lab 1a, Class Programming And Testing` `...`	`// Lab 1a, Class Programming And Testing` `...`	`// Lab 1a, Class Programming And Testing` `...`

```
#ifndef Road_h
#define Road_h

class Road
{
  ...
};

#endif
```

```
#include "Road.h"

...
```

```
#include <iostream>
...
using namespace std;

#include <cassert>
...

#include "Road.h"
#include "Road.h" // testing ifndef

int main()
{
  // print my name and this assignment's title
  ...

  Road road;
  ...test all getters and setters

  // object copy testing
  {
    const Road copy = road; // a read-only copy
    ...confirm that copy's contents match road's
  }

  // object assignment testing
  {
    Road copy; copy = road;
    ...confirm that copy's contents match road's
  }
}
```

LAB 1b: More Class Programming And Testing `[ Time.h, Time.cpp, and TimeDriver.cpp ]`
Purpose. This lab provides more class programming and testing experience for you. But in this case, the names of the interface functions are specified.

Requirements. Write 3 files: **`Time.h`, `Time.cpp`, and `TimeDriver.cpp`**, to solve exercise #19 on page 24 of the Childs textbook. Include as much testing in the driver CPP as you feel is necessary to assure that your class works right.

Do NOT add functions that are not specified -- no construtors, no "getHour", etc. Use the only getters that you have to do testing. That is, set hours, minutes, and seconds, determine what getTimeInHours should return, and test that.

DO NOT write getHour, getMinute, or getSecond.

Be sure to identify member functions as either getters or setters. If a function does not mutate any of the data members, then tag it with the "trailing `const`.

Submit the three files (2 CPPs and 1 H) to the class website for credit. Do NOT submit OBJ or EXE or any other executable file. And do NOT use file I/O -- all your input should be hard-coded, and all your output should be to `cout`.

Program I/O. Input: All hard-coded in the driver CPP only. (If you have a `cin` statement, then you did not do this right!) Output: From the driver CPP only, console (`cout`) output only.

Example. (as in lab 1a)

How to pause a console program: *show / hide*

GRADING RUBRIC *show/hide*
